Patent Application
Docket Number: EMC-02-119-CIP3
Applicants: Glade et al.
EMC CONFIDENTIAL
Express Mail Label No. EK900599093US

EMC-02-119-CIP3 PATENT

APPLICATION FOR UNITED STATES PATENT

Title: System and Method for Managing Storage Networks and for Handling Errors in such a Network

By: Bradford B. Glade, Fernando Oliveira, Jeffrey A. Brown, Peter J. McCann, David Harvey, James A. Wentworth, III, Walter M. Caritj, Matthew Waxman, and Lee W. VanTine

Related Applications

This application is a continuation-in-part of U.S. Patent Application Serial No. 09/608,521 entitled "Method and Apparatus for Implementing High-Performance Scaleable Data Processing and Storage Systems" filed on June 30, 2000, and which is hereby incorporated in its entirety by this reference. This application is related to co-pending U.S. Patent Application Serial No. To Be Determined filed on even date with this application and entitled "System and Method for Managing Storage Networks and Providing Virtualization of Resources in such a Network" and which is assigned to the same assignee as this application. Additionally, this application is related to co-pending U.S. Patent Application Serial No. To Be Determined filed on even date with this application and entitled "System and Method for Managing Storage Networks and for Managing Scalability of Volumes in such a Network" and which is assigned to the same assignee as this application.

Field of the Invention

This invention relates generally to managing and analyzing data in a data storage environment, and more particularly to a system and method for managing physical and logical components of storage area networks.

Background of the Invention

Computer systems are constantly improving in terms of speed, reliability, and processing capability. As is known in the art, computer systems which process and store large amounts of data typically include a one or more processors in communication with a shared data storage system in which the data is stored. The data storage system may include one or more storage devices, usually of a fairly robust nature and useful for storage spanning various temporal requirements, e.g., disk drives. The one or more processors perform their respective operations using the storage system. Mass storage systems (MSS) typically include an array of a plurality of disks with on-board intelligent and communications electronics and software for making the data on the disks available.

To leverage the value of MSS, these are typically networked in some fashion, Popular implementations of networks for MSS include network attached storage (NAS) and storage area networks (SAN). In NAS, MSS is typically accessed over known TCP/IP lines such as Ethernet using industry standard file sharing protocols like NFS,

HTTP, and Windows Networking. In SAN, the MSS is typically directly accessed over Fibre Channel switching fabric using encapsulated SCSI protocols.

Each network type has its advantages and disadvantages, but SAN's are particularly noted for providing the advantage of being reliable, maintainable, and being a scalable infrastructure but their complexity and disparate nature makes them difficult to centrally manage. Thus, a problem encountered in the implementation of SAN's is that the dispersion of resources tends to create an unwieldy and complicated data storage environment. Reducing the complexity by allowing unified management of the environment instead of treating as a disparate entity would be advancement in the data storage computer-related arts. While it is an advantage to distribute intelligence over various networks, it should be balanced against the need for unified and centralized management that can grow or scale proportionally with the growth of what is being managed. This is becoming increasingly important as the amount of information being handled and stored grows geometrically over short time periods and such environments add new applications, servers, and networks also at a rapid pace.

Summary of the Invention

To overcome the problems described above and to provide the advantages also described above, the present invention in one embodiment includes a method for consistent error presentation within a system of one or more storage area networks including an intelligent multi-protocol switch (IMPS) combined with a storage and switch controller including at least one microprocessor and a disk array for storing meta-data related to the plurality of data storage volumes such that the one or more data storage networks are managed by the controller using the meta-data and by interacting with the IMPS. And the method comprises the steps of, in response to receiving an error from a data storage system in one of the storage area networks, the controller processing the error by selectively either masking the error from the host or presenting the error to the host as being from the controller rather than the data storage system.

In another embodiment a system is configured for carrying out such method steps. In another embodiment, a program product includes a computer-readable medium having code included on the medium configured to carry out computer-executed steps that are similar or identical to those described above with reference to the embodiment of the method.

Brief Description of the Drawings

The above and further advantages of the present invention may be better under stood by referring to the following description taken into conjunction with the accompanying drawings in which:

Fig. 1 is a block diagram showing a Data Storage environment including a new architecture embodying the present invention and which is useful in such an environment;

Fig. 2 is another block diagram showing hardware components of the architecture shown in Fig. 1;

Fig. 3 is another block diagram showing hardware components of a processor included in the architecture and components of respective Figs. 1 and 2;

Fig. 4 is another block diagram showing hardware components of a disk array included in the architecture and components of respective Figs. 1 and 2;

Fig. 5 is a schematic illustration of the architecture and environment of Fig. 1;

Fig. 6 is a functional block diagram showing software components of the processor shown in Fig. 3;

Fig. 7 is a functional block diagram showing software components of intelligent switches which are included in the architecture of Fig. 1 and which are also shown in the hardware components of Fig. 2;

Fig. 8 shows an example of implementation of clones in the environment of Fig. 1;

Fig. 9 shows an example of SNAP Processing at a time in the environment of Fig. 1;

Fig. 10 shows another example of SNAP Processing at another time in the environment of Fig. 1;

Fig. 11 shows a schematic block diagram of software components of the architecture of Fig. 1 showing location and relationships of such components to each other;

Fig. 12 shows an example of Virtualization Mapping from Logical Volume to Physical Storage employed in the Data Storage Environment of Fig. 1;

Fig. 13 shows an example of SNAP Processing employing another example of the Virtualization Mapping and showing before a SNAP occurs;

Fig. 14 shows another example of SNAP Processing employing the Virtualization Mapping of Fig. 12 and showing after a SNAPSHOT occurs but BEFORE a WRITE has taken place;

Fig. 15 shows another example of SNAP Processing employing the Virtualization Mapping of Fig. 12 and showing after a SNAPSHOT occurs and AFTER a WRITE has taken place;

Fig. 16 is a flow logic diagram illustrating a method of managing the resources involved in the SNAP Processing shown in Figs. 14-15;

Fig. 17 is another flow logic diagram illustrating a method of managing the resources involved in the SNAP Processing shown in Figs. 14-15;

Fig. 18 is an example of a data structure involved in a process of identifying and handling volumes that have extent maps that have become fragmented during SNAP Processing and a process to reduce such fragmentation;

Fig. 19 is another example of a data structure involved in a process of identifying and handling volumes that have extent maps that have become fragmented during SNAP Processing and a process to reduce such fragmentation;

Fig. 20 is another example of a data structure involved in a process of identifying and handling volumes that have extent maps that have become fragmented during SNAP Processing and a process to reduce such fragmentation;

Fig. 21 is a schematic showing a hierarchical structure employed in the Data Storage Environment of Fig. 1 within the storage processor of Fig. 3 for allowing storage applications to be managed for consistent error presentation or handling;

Fig. 22 is a schematic of associations present when handling errors for consistent error presentation or handling with the hierarchical structure of Fig. 21;

Fig. 23 is an example of a structure implementing error handling for consistent error presentation or handling, and which is a simplified version of the type of structure shown in Fig. 21;

Fig. 24 show method steps for consistent error presentation or handling and using the example structure of Fig. 23;

Fig. 25 show additional method steps for consistent error presentation or handling and using the example structure of Fig. 23; and

Fig. 26 shows a software application for carrying out the methodology described herein and a computer medium including software described herein.

Detailed Description of the Preferred Embodiment

The methods and apparatus of the present invention are intended for use in Storage Area Networks (SAN's) that include data storage systems, such as the Symmetrix Integrated Cache Disk Array system or the Clariion Disk Array system available from EMC Corporation of Hopkinton, MA and those provided by vendors other than EMC.

The methods and apparatus of this invention may take the form, at least partially, of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, random access or read only-memory, or any other machine-readable storage medium, including transmission medium. When the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. The methods and apparatus of the present invention may be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via any other form of transmission. And may be implemented such that herein, when the program code is received and loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. When implemented on a general-purpose processor, the program code combines with the processor to provide a

unique apparatus that operates analogously to specific logic circuits. The program code (software-based logic) for carrying out the method is embodied as part of the system described below.

Overview

The embodiment of the present invention denominated as FabricX architecture allows storage administrators to manage the components of their SAN infrastructure without interrupting the services they provide to their clients. This provides for a centralization of management allowing the storage infrastructure to be managed without requiring host-based software or resources for this management. For example, data storage volumes can be restructured and moved across storage devices on the SAN while the hosts accessing these volumes continue to operate undisturbed.

The new architecture also allows for management of resources to be moved off of storage arrays themselves, allowing for more centralized management of heterogeneous data storage environments. Advantages provided include: (1) centralized management of a storage infrastructure; (2) storage consolidation and economical use of resources; (3) common replication and mobility solutions (e.g., migration) across heterogeneous storage subsystems; and (4) storage management that is non-disruptive to hosts and storage subsystems

Architecture

Referring now to Fig. 1, reference is now made to a data storage environment 10 including an architecture including the elements of the front-end storage area network 20 and a plurality of hosts 1-N shown as hosts 13, 14, and 18, wherein some hosts may communicate through the SAN and others may communicate in a direct connect fashion, as shown. The architecture includes two intelligent multi-protocol switches (IMPS's) 22 and 24 and storage and switch controller 26 to form a combination 27 which may also be denominated as a FabricX Instance 27. In communication with the Instance through an IP Network 64 and management interface 43 is an element management station (EMS) 29, and back-end storage network 42. Such back-end storage may include one or more storage systems, such as the EMC Clariion and Symmetrix data storage systems from EMC of Hopkinton, MA.

Generally such a data storage system includes a system memory and sets or pluralities and of multiple data storage devices or data stores. The system memory can comprise a buffer or cache memory; the storage devices in the pluralities and can comprise disk storage devices, optical storage devices and the like. However, in a preferred embodiment the storage devices are disk storage devices. The sets represent an array of storage devices in any of a variety of known configurations. In such a data storage system, a computer or host adapter provides communications between a host system and the system memory and disk adapters and provides pathways between the system memory and the storage device pluralities. Regarding terminology related to the

preferred data storage system, the host or host network is sometimes referred to as the front end and from the disk adapters toward the disks is sometimes referred to as the back end. Since the invention includes the ability to virtualize disks using LUNs as described below, a virtual initiator may be interchanged with disk adapters. A bus interconnects the system memory, and communicates with front and back end. As will be described below, providing such a bus with switches provides discrete access to components of the system.

Referring again to Figure 1, the Data Storage Environment 10 provides architecture in a preferred embodiment that includes what has been described above as a FabricX Instance. Pairs of the IMPS switch are provided for redundancy; however, one skilled in the art will recognize that more or less switches and processors could be provided without limiting the invention and that the Controller could also be provided in redundancy. Storage from various storage subsystems is connected to a specific set of ports on an IMPS. As illustrated, the imported storage assets and these back-end ports make up the Back-End SAN 41 with a networked plurality of data storage arrays 38, and 40, and which also may be directly connected to either IMPS, as shown with arrays 30-34 so connected to the Instance 27 through IMPS 24, but although not shown could also be connected directly to the Storage and Switch Controller.

It is known in SAN networks using Fibre Channel and/or SCSI protocols that such data devices as those represented by disks or storage 30-40 can be mapped using a protocol to a Fibre Channel logical unit (LUN) that act as virtual disks that may be presented for access to one or more hosts, such as hosts 13-18 for I/O operations. LUN's

are also sometimes referred to interchangeably with data volumes which at a logical level represent physical storage such as that on storage 30-40.

Over the referred IP Network 64 and by communicating through the management interface 43, a Storage Administrator using the EMS 29 may create virtual LUN's (Disks) that are composed of elements from the back-end storage. These virtual devices which may be represented, for example by a disk icon (not shown) grouped with the intelligent switch, are made available through targets created on a specific set of intelligent switch ports. Client host systems connect to these 'front-end' ports to access the created volumes. The client host systems, the front-end ports, and the virtual LUN's all form part of the Front-End SAN 20. Note Hosts, such as Host 13 may connect directly to the IMPS.

The combined processing and intelligence of the switch and the FabricX Controller provide the connection between the client hosts in the front-end SAN and the storage in the back-end SAN. The FabricX Controller runs storage applications that are presented to the client hosts. These include the Volume Management, Data Mobility, Snapshots, Clones, and Mirrors, which are terms of art known with EMC's Clariion data storage system. In a preferred embodiment the FabricX Controller implementation is based on the CLARiiON Barracuda storage processor and the CLARiiON Flare software implementation which includes layered drivers that are discussed below.

Hardware Components

Referring to Fig. 2, hardware components of the architecture in the environment shown in Fig. 1 are now described in detail. A FabricX instance 27 is comprised of several discrete hardware subsystems that are networked together. The major subsystems include a Control Path Processor (CPP) 58 and a Disk Array Enclosure (DAE) 54, each described in more detail in Figs. 3 and 4.

The CPP provides support for storage and switch software applications and runs the software that handles exceptions that occur on the fast-path. Regarding where software runs, in the exemplary embodiment, software for management by the Storage and Switch Controller is shown running on the CPP; however, that is merely an example and any or all software may be loaded and run from the IMPS or anywhere in the networked environment. Additionally the CPP supports management interfaces used to configure and control the instance. The CPP is composed of redundant storage processors and is further described with reference to Fig. 3.

The DAE, together with the disks that it contains provide the persistent storage of the meta-data for the FabricX instance. The meta data includes configuration information that identifies the components of the instance, for example, the identities of the intelligent switches that make up the instance, data describing the set of exported virtual volumes, the software for the Controller, information describing what hosts and initiators are allowed to see what volumes, etc. The DAE is further described with reference to Fig. 4. The IMPS 22 or 24 provide storage virtualization processing in the data-path (also known

as fast-path processing), and pass control to the CPP when exceptions occur for requests that it cannot handle.

Referring to Fig. 2, each FabricX instance may be managed by an administrator or user using EMS 29. Preferably, a given EMS is capable of managing one or more FabricX instances and communicates to the FabricX instance components through one or more IP networks.

Referring to Fig. 3, CPP 58 preferably includes two storage processors (SP's) 72 and 74, which may be two Intel Pentium IV microprocessors or similar. The two storage processors in the CPP communicate with each other via links 71, which may be for example redundant 2 Gbps Fibre Channel links, each provided in communication with the mid-plane 76. Each CPP contains fan modules 80 that connect directly to the mid-plane 76. The CPP contains two power supplies 78 and 82 (Power Supply A and B). In a preferred embodiment, the power supplies are redundant, have their own line cord, power switch, and status light, and each power supply is capable of providing full power to the CPP and its DAE. During normal operation the power supplies share load current. These redundant standby power supplies provide backup power to the CPP to ensure safety and integrity of the persistent meta-data maintained by the CPP.

Referring to Fig. 4, the DAE 54 is shown. A FabricX instance 27 preferably has a single DAE 54, which is loaded with four disk drives 100 (the number of drives is a variable choice, however). These disk drives provide the persistent storage for meta-data of the instance, wherein the meta-data is used for certain management and control functions. None of this storage is directly accessible or visible to hosts on the front-end.

The meta-data on the disk drives is three-way mirrored to provide protection from disk failures. Each SP has a single arbitrated loop that provides its connection to the DAE. Each Link Control Card or LCC 98 and 102 connects the FabricX SP's to the meta-data storage devices or disk drives within the Disk Array Enclosure.

Fig. 5 shows a schematic illustration of the architecture and environment of Fig. 1 in detail with preferred connectivity and in a preferred two IMPS configuration (IMPS 22 and IMPS 24). Host Systems 13-18 preferably communicate with FabricX via a SCSI protocol running over Fibre Channel. Each Fibre Channel port of each IMPS is distinguished as being either a front-end port, a back-end port, a control-port, or an inter-switch port. Hosts connect to the FabricX instance 27 via front-end ports. Front-end ports support SCSI targets and preferably have virtualizing hardware to make up an intelligent port. The host's connection to the port may be direct as in the case of labeled Host 1 or indirect such as Host 2 via layer 2 Fibre Channel switches such as Switch 60-SW1 and Switch 62-SW2. Hosts may establish multiple paths to their storage by connecting to two or more separate front-end ports for high availability and performance; however, the preferred FabricX instance architecture allows hosts to be configured with a single path for the sake of simplicity. In some configurations, not shown for simplicity, the switches 60-SW1 and 62-SW2 could be combined and/or integrated with the IMPS without departing from the spirit of the invention.

An IMPS can be used to support virtual SAN's (VSAN's), to parse between front-end SAN's and back-end SAN's even if such SAN's are not physically configured. In general, switches that support VSANs allow a shared storage area network to be

configured into separate logical SANs providing isolation between the components of different VSANs. The IMPS itself may be configured in accordance with specifications from such known switch vendors as Brocade and Cisco.

Each intelligent switch preferably contains a collection of SCSI ports, such as Fibre Channel, with translation processing functions that allow a port or associate hardware to make various transformations on the SCSI command stream flowing through that port. These transformations are performed at wire-speeds and hence have little impact on the latency of the command. However, intelligent ports are only able to make translations on read and write commands. For other SCSI commands, the port blocks the request and passes control for the request to a higher-level control function. This process is referred to as faulting the request. Faulting also occurs for read and write commands when certain conditions exist within the port. For example, a common transformation performed by an intelligent port is to map the data region of a virtual volume presented to a host to the data regions of back-end storage elements. To support this, the port maintains data that allows it to translate (map) logical block addresses of the virtual volume to logical back-end addresses on the back-end devices. If this data is not present in the port when a read or write is received, the port will fault the request to the control function. This is referred to as a map fault.

Once the control function receives a faulted request it takes whatever actions necessary to respond to the request (for example it might load missing map data), then either responds directly to the request or resumes it. The control function supported may be implemented differently on different switches. On some vendor's switches the control

function is known to be supported by a processor embedded within the blade containing the intelligent ports, on others it is known to provide it as an adjunct processor which is accessed via the backplane of the switch, a third known configuration is to support the control function as a completely independent hardware component that is accessed through a network such as Fibre Channel or IP.

Back-end storage devices connect to FabricX via the Fibre Channel ports of the IMPSs that have been identified as back-end ports (oriented in Fig. 5 toward the back-end SAN). Intelligent ports act as SCSI initiators and the switch routes SCSI traffic to the back-end targets 103-110 respectively labeled T1-TN through the back-end ports of the respective IMPS's. The back-end devices may connect directly to a back-end IMPS if there is an available port as shown by T5, or they may connect indirectly such as in the case of T1 via a layer 2 Fibre Channel switch, such as Switch 60-SW3, and Switch 62-SW4.

The EMS 29 connects to FabricX in a preferred embodiment through an IP network, e.g. an Ethernet network which may be accessed redundantly. The FabricX CPP 58 in a preferred embodiment has two 10/100 Mbps Ethernet NIC that is used both for connectivity to the IMPS (so that it can manage the IMPS and receive SNMP traps), and for connectivity to the EMS. It is recommended that the IP networks 624a-b provided isolation and dedicated 100 Mbps bandwidth to the IMPS and CPP.

The EMS in a preferred embodiment is configured with IP addresses for each Processor 72-74 in the FabricX CPP. This allows direct connection to each processor. Each Processor preferably has its own Fibre Channel link that provides the physical path

to each IMPS in the FabricX instance. Other connections may also work, such as the use of Gigabit Ethernet control path connections between the CPP and IMPS. A logical control path is established between each Processor of the CPP and each IMPS. The control paths to IMPS's are multiplexed over the physical link that connects the respective SP of the CPP to its corresponding IMPS. The IMPS provides the internal routing necessary to send and deliver Fiber Channel frames between the SP of the CPP and the respective IMPS. Other embodiments are conceivable that could use IP connectivity for the control path. In such a case the IMPS could contain logic to route IP packets to the SP.

Software Components

Reference is made to Figs. 6 and 7, showing a functional block diagram of software comprised in modules that run on the Storage Processors (SP) 72 or 74 within Control Path Processor (CPP) 58 and on the IMPS owned by the instance. Each of these storage processors operates as a digital computer preferably running Microsoft Window XP Embedded and hosts software components. Software Components of each SP are now described.

The CPP-based software includes a mixture of User-Level Services 122 and Kernel-mode or Kernel services 128. The Kernel services include Layered Drivers 123,

Common Services 125, Switch Independent Layer (SIL) 126, and Switch Abstraction Layer-Control Path Processor (SAL-CPP) 127. The IMPS-based software preferably runs on a control processor within the vendor's switch. This processor may be embedded on an I/O blade within the switch or implemented as a separate hardware module.

The SAL-CPP 127 provides a vendor-independent interface to the services provided by the IMPS's that form a FabricX instance. This software layer creates and manages a IMPS Client for each IMPS that is part of the FabricX instance. The following services are provided by the SAL-CPP. There is a Switch Configuration and Management Services (SWITCH CONFIG & MGMT) in the SAL-CPP that provides uniform support for configuring the IMPS, zone configuration and management, name service configuration and management, discovering the ports supported by the IMPS, reporting management related events such as Registered State Change Notifications (RSCNs), and component failure notifications. The service interfaces combined with the interfaces provided by the user-level Switch Management service encapsulate the differences between different switch vendors and provide a single uniform interface to the FabricX management system. The Switch Adapter Port Driver (SAPD) of the Kernel Services 128 uses these interfaces to learn what ports are supported by the instance so that it can create the appropriate device objects representing these ports.

Referring to Fig. 6, the SAL-CPP 127 provides Front-End Services (FRONT-END SVCS) that include creating and destroying virtual targets, activate and deactivate virtual targets. Activation causes the target to log into the network while deactivation causes it to log out. Storage Presentation objects represent the presentation of a volume

on a particular target. These Front-End Services also include LUN mapping and/or masking on a per initiator and per target basis.

Referring again to Fig. 6, the SAL-CPP 127 provides Back-End Services (BACK-END SVCS) that include discovering back-end paths and support for storage devices or elements, including creating and destroying objects representing such devices or elements. Back-End Services include managing paths to the devices and SCSI command support. These services are used by FlareX of the Layered Drivers 123 to discover the back-end devices and make them available for use, and by the Path Management of the Switch Independent Layer (SIL). The SIL is a collection of higher-level switch-oriented services including managing connectivity to storage devices. These services are implemented using the lower-level services provided by the SAL-CPP.

SAL-CPP 127 provides a volume management (Volume MGMT) service interface that supports creating and destroying virtual volumes, associating virtual volumes with back-end Storage Elements, and composing virtual volumes for the purpose of aggregation, striping, mirroring, and/or slicing. The volume management service interface also can be used for loading all or part of the translation map for a volume to a virtualizer, quiescing and resuming IO to a virtual volume, creating and destroying permission maps for a volume, and handling map cache miss faults, permission map faults, and other back-end errors. These services are used by the volume graph manager (VGM) in each SP to maintain the mapping from the virtual targets presented out the logical front of the instance to the Storage Elements on the back-end.

There are other SAL-CPP modules. The SAL copy service (COPY SVCS) functions provide the ability to copy blocks of data from one virtual volume to another. The Event Dispatcher is responsible for delivering events produced from the IMPS to the registered kernel-based services such as Path Management, VGM, Switch Manager, etc.

The Switch and Configuration Management Interface is responsible for managing the connection to an IMPS. Each Storage Processor maintains one IMPS client for each IMPS that is part of the instance. These clients are created when the Switch Manager process directs the SAL-CPP to create a session with a IMPS.

The Switch Independent Layer (SIL) 126 is a collection of higher-level switch-oriented services. These services are implemented using the lower-level services provided by the SAL-CPP. These services include:

- Volume Graph Manager (VGM) – The volume graph manager is responsible for processing map-miss faults, permission map faults, and back-end IO errors that it receives from the SAL-CPP. The VGM maintains volume graphs that provide the complete mapping of the data areas of front-end virtual volumes to the data areas of back-end volumes. The Volume Graph Manager provides its service via a kernel DLL running within the SP.

- Data Copy Session Manager – The Data Copy Session Manager provides high-level copy services to its clients. Using this service, clients can create sessions to control the copying of data from one virtual volume to another. The service

allows its clients to control the amount of data copied in a transaction, the amount of time between transactions, sessions can be suspended, resumed, and aborted. This service builds on top of capabilities provided by the SAL-CPP's Data Copy Services. The Data Copy Session Manager provides its service as a kernel level DLL running within the SP.

- Path Management – The path management component of the SIL is a kernel-level DLL that works in conjunction with the Path Manager. Its primary responsibility is to provide the Path Manager with access to the path management capabilities of the SAL-CPP. It registers for path change events with the SAL-CPP and delivers these events to the Path Manager running in user-mode. Note, in some embodiments, the Path Management, or any of the other services may be configured to operate elsewhere, such as being part of another driver, such as FlareX.

- Switch Management – The switch management component of the SIL is a kernel-level DLL that works in conjunction with the Switch Manager. Its primary responsibility is to provide the Switch Manager with access to the switch management capabilities of the SAL-CPP.

The CPP also hosts a collection of Common Services 125 that are used by the layered application drivers. These services include:

- Persistent Storage Mechanism (PSM) – This service provides a reliable persistent data storage abstraction. It is used by the layered applications for storing their meta-data. The PSM uses storage volumes provided by FlareX that are located on the Disk Array Enclosure attached to the CPP. This storage is accessible to both SPs, and provides the persistency required to perform recovery actions for failures that occur. Flare provides data-protection to these volumes using three-way mirroring. These volumes are private to a FabricX instance and are not visible to external hosts.

- Distributed Lock Service (DLS) – This service provides a distributed lock abstraction to clients running on the SPs. The service allows clients running on either SP to acquire and release shared locks and ensures that at most one client has ownership of a given lock at a time. Clients use this abstraction to ensure exclusive access to shared resources such as meta-data regions managed by the PSM. Message Passing Service (MPS) – This service provides two-way communication sessions, called filaments, to clients running on the SPs. The service is built on top of the CMI service and adds dynamic session creation to the capabilities provided by CMI. MPS provides communication support to kernel-mode drivers as well as user-level applications.

- Communication Manager Interface (CMI) – CMI provides a simple two-way message passing transport to its clients. CMI manages multiple communication

paths between the SPs and masks communication failures on these. The CMI transport is built on top of the SCSI protocol which runs over 2Gbps Fibre-Channel links that connect the SPs via the mid-plane of the storage processor enclosure. CMI clients receive a reliable and fast message passing abstraction. CMI also supports communication between SPs within different instances of FabricX. This capability will be used to support mirroring data between instances of FabricX.

The CPP includes Admin Libraries that provide the management software of FabricX with access to the functionality provided by the layered drivers such as the ability to create a mirrored volume or a snapshot. The Admin Libraries, one per managed layer, provide an interface running in user space to communicate with the managed layers. The CPP further includes Layered Drivers 123 providing functionality as described below for drivers denominated as Flare, FlareX (FLARE_X), Fusion, Clone/Mirror, PIT Copy, TDD, TCD, and SAPD.

Flare provides the low-level disk management support for FabricX. It is responsible for managing the local Disk Array Enclosure used for storing the data of the PSM, the operating system and FabricX software, and initial system configuration images, packages, and the like. It provides the RAID algorithms to store this data redundantly.

The FlareX component is responsible for discovering and managing the back-end storage that is consumed by the FabricX instance. It identifies what storage is available,

the different paths to these Storage Elements, presents the Storage Elements to the management system and allows the system administrator to identify which Storage Elements belong to the instance. Additionally, FlareX may provide Path Management support to the system, rather than that service being provided by the SIL as shown. In such a case, FlareX would be responsible for establishing and managing the set of paths to the back-end devices consumed by a FabricX instance. And in such a care it would receive path related events from the Back-End Services of the SAL-CPP and responds to these events by, for example, activating new paths, reporting errors, or updating the state of a path.

The Fusion layered driver provides support for re-striping data across volumes and uses the capabilities of the IMPS have to implement striped and concatenated volumes. For striping, the Fusion layer (also known as the Aggregate layer), allows the storage administrator to identify a collection of volumes (identified by LUN) over which data for a new volume is striped. The number of volumes identified by the administrator determines the number of columns in the stripe set. Fusion then creates a new virtual volume that encapsulates the lower layer stripe set and presents a single volume to the layers above.

Fusion's support for volume concatenation works in a similar way; the administrator identifies a collection of volumes to concatenate together to form a larger volume. The new larger volume aggregates these lower layer volumes together and presents a single volume to the layers above. The Fusion layer supports the creation of many such striped and concatenated volumes.

Because of its unique location in the SAN infrastructure, FabricX, can implement a truly non-disruptive migration of the dataset by using the Data Mobility layer driver that is part of the Drivers 123. The client host can continue to access the virtual volume through its defined address, while FabricX moves the data and updates the volume mapping to point to the new location.

The Clone driver provides the ability to clone volumes by synchronizing the data in a source volume with one or more clone volumes. Once the data is consistent between the source and a clone, the clone is kept up-to-date with the changes made to the source by using mirroring capabilities provided by the IMPS's. Clone volumes are owned by the same FabricX instance as the source; their storage comes from the back-end Storage Elements that support the instance.

The Mirror driver supports a similar function to the clone driver however, mirrors are replicated between instances of FabricX. The mirror layered driver works in conjunction with the mirror driver in another instance of FabricX. This application provides the ability to replicate a source volume on a remote FabricX instance and keep the mirror volume in synch with the source.

The PIT (Point-In-Time) Copy driver, also known as Snap, provides the ability to create a snapshot of a volume. The snapshot logically preserves the contents of the source volume at the time the snapshot is taken. Snapshots are useful for supporting non-intrusive data backups, replicas for testing, checkpoints of a volume, and other similar uses.

The Target Class Driver and Target Disk Driver (TCD/TDD) layer provides SCSI Target support. In FabricX these drivers mostly handle non-read and write SCSI commands (such as INQUIRY, REPORT_LUNS, etc). The drivers are also responsible for error handling, when errors cannot be masked by the driver layers below, the TCD/TDD is responsible for creating the SCSI error response to send back to the host. The TCD/TDD Layer also implements support for the preferred CLARiiON functionality which provides the means of identifying what LUNs each initiator should see. This is known as LUN masking. The feature also provides for LUN mapping whereby the host visible LUN is translated to an instance-based LUN. Additionally such functionality when combined with a host agent provides the ability to identify which initiators belong to a host to simplify the provisioning of LUN masking and mapping.

The Switch Adapter Port Driver (SAPD) is presented as a Fibre-Channel Port Driver to the TCD/TDD (Target Class Driver/Target Disk Driver) drivers, but rather than interfacing with a physical port device on the SP, the driver interfaces with the SAL-CPP and creates a device object for each front-end port of each IMPS that is part of the FabricX instance. The SAPD registers with the SAL-CPP to receive non-IO SCSI commands that arrive. The SAL-CPP will deliver all non-IO SCSI commands received for LUs owned by this driver's SP to this SAPD. The SAPD runs as a kernel-mode driver.

The following services are user based: Governor and Switch Management. The Governor is an NT Service that is responsible for starting other user-level processes and monitoring their health and restarting them upon failure. The Switch Manager controls

sessions created in the SAL-CPP for each IMPS. The Switch Manager is responsible for establishing the connections to the switches under its control and for monitoring and reacting to changes in their health. Each SP hosts a single Switch Manager that runs as a User-level process and a Kernel-mode service within the SP.

Reference is made once again to Fig. 6. The Raid++ services encapsulates the legacy logic dealing with the configuration and management of the array logical components (such as storage group, LUNs, etc.) and physical components (such as cabinets, DAEs, disk drives, etc.). The Providers are plug-in modules to the CIMOM which provide the functionality for a particular set of managed objects. Providers represent the objects (class definitions and behaviors) as defined in the object model of the managed element. The Admin Libraries include an interface between the user space management tasks and the kernel mode drivers' instrumenting the management of FabricX. The Admin Libraries accept requests using a Tagged Length Data (TLD) self-describing message format from the management layer and converts those requests into the specific IOCTL calls required to realize those requests. Responses are returned using the same format.

The Path Management is responsible for the construction and management of paths to back-end Storage Elements and is part of Kernel-mode services. It notes when paths change state; based on these state changes it applies its path management policies to take any adjusting actions. For example, upon receiving a path failure notification, the Path Management might activate a new path to continue the level of service provided for a back-end volume.

One function of FabricX Volume Management is to combine elements of the physical storage assets of the FabricX Instance into logical devices. The initial implementation of the FabricX Volume Manager is based on the Flare Fusion Driver. As in Flare, the basic building blocks of the volumes exported by FabricX are constructed from the back-end storage devices. Each device visible to the FabricX instance will be initially represented as an un-imported Storage Element. The storage administrator will able to bind the individual storage elements into single disk RAID Groups. From these RAID Groups the administrator can define Flare Logical Units (FLU). In the FabricX environment the FLUs will be exported by the FlareX component to the layered drivers above.

Flare Fusion imports FLUs and aggregates them into Aggregate Logical Units (ALU). When a logical unit or SCSI Disk is presented to a client host it is called a Host Logical Unit (HLU). HLUs can be created by: directly exporting a FLU; exporting an ALU created by concatenating two or more FLUs; and exporting an ALU created by striping two or more FLUs.

The FabricX Inter Process Communication Transport (FIT) provides the message passing support necessary for the SAL Agents running on the IMPS's to communicate with the SAL-CPP client instance running on each SP. This transport provides a model of asynchronous communication to its clients and is responsible for monitoring and reporting on the health of the communications network connecting the IMPSs to the CPPs. FIT uses a proprietary protocol developed on top of the SCSI/FC protocol stack to provide a control path between the SPs and the IMPS's. This protocol runs over the

Fibre Channel connecting the SP to the IMPS's switch. FIT supports multiple transport protocols. In addition to SCSI/FC FIT also supports TCP/IP.

Fig. 7 shows the software components of the intelligent switch or IMPS 22 or 24. IMPS Software Components include an IMPS API and FIT 132, SAL Agent 130, and IMPS Operating System 134. Each switch vendor provides a software interface to the services provided by their switch that is the IMPS API and the SP's communicate with this API. This API provides support to application software that runs within the switch or blade and in some cases this interface is remote to the switch. FabricX isolates the storage application logic from these differences by defining the Switch Abstraction Layer (SAL-CPP) discussed with reference to Fig. 6. The SAL Agent 130 is an application from the perspective of the IMPS and works with SAL-CPP 127 (Fig. 6) and the FIT of the SAL-CPP and of the IMPS. The Agent 130 directly uses the native IMPS API to control and communicate with the switch. Its function is to provide access to the services of the switch to the Control Path Processors. The IMPS operating system varies from switch vendor to vendor. For example, Cisco's Intelligent Switches use Monte Vista Linux, while the Brocade switches use NetBSD.

FabricX preferably uses the Flare architecture from CLARiiON. This architecture uses a Windows driver stack. At the lowest layer is the code, labeled FlareX (Flare_X) that interfaces to the back-end devices. The storage applications are logically layered above FlareX as Windows device drivers, including Fusion, Mirror_View, Snap_View, Clone_View, and TCD/TDD. These layers provide support for features such as volume

concatenation, striping, clones, snapshots, mirrors, and data migration. These layers also define virtualized Logical Unit objects.

This architecture includes replication layers: Snap_View, Clone_View, and Mirror_View are layered above the Fusion layer of code and consume the logical units (volumes) presented by the Fusion layer, they likewise present logical units to the layers above. The replication layers have no knowledge of back-end devices or data placement thereon.

The inventors have critically recognized that prior art storage arrays had limited processes to create clones, which are replicas stored within an array, and mirrors, which are replicas stored on different arrays. This is because the front end of an array has no way to directly access to a back end device in some other array. Remote mirroring in such prior art configurations is processed through two different arrays, one attached to a host and one attached to remote back end devices. These two arrays are then connected to each other via a WAN. However, the present invention being based on a switch and storage architecture does not have such limitations. Instead, all back end devices are equivalent irregardless of their physical location though latency of access may vary. A back end device may be connected to a back-end port of a switch through some WAN making it physically remote. With a switch-based architecture clones, which are replicas stored within the storage managed by the instance, can be created from storage that is physically remote from the Controller and switch hardware just as easily as creating it from storage which is physically close to this hardware. Only one FabricX instance is necessary, to create clones whether on physically local back-end devices or on physically

remote back-end devices. However, if it is desirable to create replicas, mirrors, between instances of FabricX that is possible. For example, one might want to create a replica on another instance for the purpose of increasing data protection and providing a means to tolerate a failure of a FabricX instance, or to replicate data using an asynchronous protocol for very long distance replication.

Further Operation Details

Fig. 8 shows an example of creation and use of clones with a FabricX instance. In this example, a Source Logical Unit (LU) 154a is connected to a host 18 together with two clones, Clone 1 152a and Clone 2 156a on FabricX Instance 27. The connection of the host is shown directly, but it may also be indirect and/or via a SAN. Clone 1 is made on a back end device provided by a locally attached array. Clone 2 is made on a remote device provided by an array located at some distance and connected to the local FabricX Instance 27 which can be accomplished by means such as Asynchronous Transfer Mode (ATM) or Dense Wave Division Multiplexing (DWDM) networks. Because these replicas are logically equivalent both are referred to as Clones. Clones made using one FabricX Instance can be accessed by hosts connected to the same FabricX Instance mapped in respective mapping areas 157 and 158 of each IMPS controlled by a respective SP. Using the mapping 157 the respective FabricX instance presents the Source LU 154a to the Host 19, while maintaining the cloned replicas152b and, via a WAN, the replica 156b related to respective arrays 160-164.

Figs. 9 and 10 show an example of Snap replicas created using FabricX. Snaps are constructed by using a process called Copy On First Write (COFW). When a Snap is created it represents the state and contents of a volume as it existed at the time of creation. With the COFW technique a snap is initially just a map of pointers 173a-c and 175a-c. These map pointers initially refer to the blocks 176a-178a in the Source, a.k.a. Ancestor, volume 179a. Only when a block in the source volume is about to change is it necessary to actually create a copy of the data as it was when the Snap was instantiated. When a write is received for a chunk of data the sub-segment(s) that refer to the chunk are copied from the Source to another area on a back-end volume (BEV) that is assigned to the Snap.

Fig. 10 shows a volume consisting of three chunks of data after the Snap is created and before any write commands are received. Both the source and Snap maps point to the same three chunks of data on back end storage, i.e., pointers 173d-f and 175d-f point to respective blocks 176b-178b on volume 179b. When a write is received, say for chunk 2, FabricX will block the write command. It will copy chunk 2 from the back end storage to some newly allocated space on back end storage and change the Snap's map to point to this new location. FabricX will then allow the write of chunk 2 to proceed to the old location. After the write is complete the mapping is as shown in Fig. 10.

Fig. 11 shows a schematic block diagram of software components of the architecture of Fig. 1 showing location and relationships of such components to each

other. The components reside within either the client (host) computers or the CPP of FabricX and use various communications method and protocols. The following describe the function of each of the major components and their interfaces to associated functions.

Several components of FabricX System Management reside on Host computers (those using the storage services of FabricX and/or those managing FabricX), these are referred to as Client Components and are shown in group 18. One component in particular, Raid++, has both a client and server instance, shown respectively in host/client group 18 or server group 210. The C++ Command Line Interface CLI 200 -- referred to here as CLI++) component resides on any system where the user wishes to manage FabricX using a text based command line interface. This component in conjunction with the Java based CLI provide the user with control over FabricX. The security model for the CLI++ is host based; the user/IP address of the host is entered into the FabricX registry to permit/limit the user's access to system management. The CLI++ uses a client instance of Raid++ to hold the model of the FabricX instance and to manipulate the configuration of FabricX. The client resident Raid++ communicates with the server based Raid++ using a messaging scheme over TCP/IP.

The Java CLI 206 provides commands that use a different management scheme from that used by the CLI++. The Java CLI captures the user command string, packages it into an XML/HTTP message and forwards it to the CIMOM on the server group 210. The CIMOM directs the command to the CLI Provider which decomposes the command and calls methods in the various CIM providers, primarily the CLARiiON provider, to effect the change.

The Java GUI 208 provides a windows based management interface. It communicates with CIMOM using standard CIM XML/HTTP protocol. The GUI effects its changes and listens for events using standard CIM XML/HTTP. The Host Agent 204 provides optional functionality by pushing down to FabricX information about the host. The following information is forwarded by the Agent explicitly to the CPP: Initiator type, Initiator options, Host device name used for push, Hostname, Host IP address, Driver name, Host Bus Adapter (HBA) model, HBA vendor string, and Host ID.

The Event Monitor 202 resides on a host and can be configured to send email, page, SNMP traps, and/or use a preferred EMC Call Home feature for service and support. The configuration is performed on the CPP and the configuration information is pushed back to the Event Monitor. The Event Monitor may also run directly on the CPP but due to memory constraints may be limited in function in comparison to running on a host computer.

Referring again to Fig. 11, the server side management components shown in group 210 interact with the user interfaces and tools for administering the system configuration and operation and to report on system operation. The server side components are comprised of middleware which resides between the user and the storage management components of the system which implement FabricX storage features. The server side components are basically divided into two groups, the legacy Raid++ module which provides the majority of the management services and the CIMOM and its providers. The Raid++ module uses a proprietary transport to communicate with management clients such as the CLI++. The Raid++ module maintains an object model

of the system that it uses for managing the system; it updates the model periodically and on demand by polling the system to rebuild the model. The CIMOM CLARiiON Provider is essentially a wrapper for the Raid++ classes and methods and translates GUI initiated CIM XML commands into calls to Raid++ to fulfill requests.

The management functions not provided by Raid++ are provided by a series of CIMOM providers which are attached to a CIMOM. The CIMOM provides common infrastructure services such as XML coding/decoding and HTTP message transport. The hosted services exclusively implemented in CIMOM providers are:

- Analyzer Provider – Provides statistics about performance of traffic on ports on the switch;
- CLI Provider – This provider implements services to allow CLI clients to access CIM managed services such as Clone, Analyzer, Fusion, and switch management;
- Clone Provider – Provides services to manage the configuration and operation of clones;
- Data Mobility Provider – Provides services to manage the configuration and operation of data migration between storage volumes transparently to the host applications using the storage;
- Fusion Provider – Provides services to configure and manage the combining of LUNs to create new LUNs of larger capacity;
- Mirror Provider – Provides services to manage the configuration and operation of mirrors; and.

- Switch Management Provider – Provides services to configure and manage the attached intelligent switch components owned by FabricX.

The above-described providers periodically poll the system infrastructure to build a model of the existing component configuration and status. If any changes are detected in configuration or status between the existing model and the newly built model, registered observers are notified of the changes. The model is then updated with the new model and saved for queries by the provider. The services of these providers can be accessed from other providers by formulating XML requests and sending them to the CIMOM. This permits providers which require the services of other providers (such as Raid++ through the CLARiiON Provider or the CIM local services such as persistent storage, directory services, or security) to access those services. Additionally Admin STL Driver Access through the server side provides access to these providers to the drivers and services of an SP as shown in group 218, including to the following drivers: Flare, Clones, Snaps, Fusion, and mirrors and services for switch management and data mobility.

Other Service Providers are shown in group 212 of the server group 210, and include the Persistent Data Service Provider, Security Provider, and Directory Service Provider. The Persistent Data Service Provider provides the CIMOM with access to the kernel-based PSM. The CIMOM uses the PSM to store meta-data relevant to the management system for example user names and passwords. The Security Provider supports authentication of users and authorization of user roles. The Directory Service Provider is used to obtain the network addresses of the systems in a domain of managed FabricX instances.

Reference will be made below to Figs. 12-20 to describe a problem solved with the architecture including the software components described above with reference to Figs. 1-11; however a general overview is now given. The inventors have critically recognized that Intelligent Multi-Protocol Switches (IMPS) 22 or 24 generally have limited memory resources available to support mapping virtual storage extents to physical storage extents. A typical switch today is capable of storing 10,000 maps per translation unit. Certain storage applications, such as a volume snapshot application can consume large numbers of these maps in support of a single volume. This further reduces the set of volumes that a translation unit can support.

This problem is addressed with the architecture of the present invention by using memory with the FabricX Storage Processor 26 or 28 to supplement the memory resources of the IMPS's translation unit and more efficiently use memory of each. The translation unit's memory resources are used to store a subset of the full set of extent maps for the volumes exported by the translation unit. Maps are loaded to the translation unit from the CPP both on demand and ahead of demand in a technique denoted as being a virtualizer application which is preferably software running in a SP or on the IMPS. In this embodiment, sequential access is detected and future requests are predicted using protection maps to mark the edges of unmapped regions. Access to a marked region in combination with the detection of sequential access triggers the preloading of additional maps prior to the arrival of the actual request.

I/O requests that arrive to find no mapped region are handled by loading the map from the control-processor. The control processor uses access data collected from the

intelligent multi-protocol switch to determine which data to replace. Supported cache replacement algorithms include least-recently used, least frequently used. The IMPS hardware is used to detect need for extents prior to access and statistical data is collected on volume access to select which cache entries to replace. The mechanism further identifies volumes whose extent maps have become fragmented and triggers a process to reduce this fragmentation.

Referring to Fig. 12, Virtualization Mapping from a logical volume 230 to physical storage 232, 234, and 236 is shown. A Host 18 write to the volume in the region 0-K is mapped to physical block segment 232 (SE1), and likewise to the volume in the region 0-(j-k) to physical block segment 234 (SE2), and also to the volume in the region 0-(n-k) to physical block segment 236. In this simple example of Virtualization Mapping the Logical Volume 240a maps byte for byte with the storage element 242a. To be precise region 0 through k-1 of 230 is mapped to SE1 (232). Likewise region k through j-1 or 230 is mapped to SE2 (234) and region j through n-1 is mapped to SE3 (236).

Referring to Fig. 13 and Fig. 14, such virtualization mapping is shown using the embodiment described above with reference to Figs. 1-11, wherein Fig. 13 shows an example case of before a Snapshot or Snap, and Fig. 14 shows after a Snap where no writes have taken place. A source logical volume 240a (Fig. 13) is mapped to segment block 242a containing data "A" in the region 0-n. At the point when the Snap is created, the Snap Cache is merely associated with this Snap volume -- the Snap volume does not actually map portions of its storage to the Snap Cache until a write takes place.

An example of solving the problem described in general is shown with reference to Fig. 15, which depicts the example of a virtualization mapping case wherein a write takes place after the Snap. This example case is described in reference to method steps shown in Flow diagrams depicted in Figs. 16 and 17. In step 300, the host 18 writes data B to region j-k of logical volume 242b (Fig. 17). In step 302 the write of data B is held by the storage application. The application carrying out the methodology is given control is step 304 which flows into the "A" connecting step 306 , which flows into its identically identified counterpart for Fig. 17.

Referring to Fig. 17, in steps 308 and 310, the application reads the original data A from region j-k of segment 242b and writes the data to the allocated storage in the snap cache 246b. In step 312, the application updates the logical mapping for the snap logical volume 244b to map region j-k to the new region for A in the snap cache 246b. Then in step 314 the application allows the original write to proceed.

Figs. 18-20 show fault extent mapping that is a further feature of the storage application. In this embodiment volumes whose extent maps have become fragmented are identified and a process to reduce this fragmentation is invoked. Referring to Fig. 18, volume 320 has been fragmented into N segments and volume 325 in conjunction with 324 represents a mechanism for preserving the image of 320. I/O operations to the region covered by 324 cause a fault which yields control of the I/O to the storage application. This simplified diagram illustrates a fault region that applies to both reads and writes however the switches and the SAL Agent and the IMPS API also support the ability to create fault-regions that apply only to reads or only to writes. In the example case, the

CPP running the storage application takes the fault and updates the extent map for the volume to map region 3 and a fault map 326 is created for the region 327 (Fig. 19). To reduce fragmentation a new fault map 330 is created and the map entries 1 and 2 are combined (Fig. 20). This causes a reduction in the number of entries required to support the volume.

Fig. 26 shows the storage software application 500 in an SP 72 or 74 and also including the software components 122-27 of Fig. 7. Computer-readable medium 502 includes program logic 503 that may include any or all of the software 122-127, and 500. Such a medium may be represented by any or all of those described at the beginning of this Detailed Description.

Figs. 21-25 describe an embodiment for ensuring consistent error presentation to hosts accessing virtualized volumes and for preserving structured error handling, and a general overview is now given. In general, it is recognized by the inventors that without this invention virtualizing intelligent switches process I/O requests from host computers at wire speed and do not present these requests to storage applications. Errors that occur while processing these requests may present an inconsistent error behavior to the hosts accessing the volume due to this errors originating from disparate back-end storage devices. Furthermore certain errors would be better handled and masked according to the structure and semantics of a virtual volume.

In this embodiment the storage application (Fig. 26) is given control of a request when exceptional conditions occur such as an error when processing the request. This invention provides a means of handling errors in a hierarchical manner allowing storage

applications to be structured in a manner consistent with the structure of a virtual volume. Errors are presented to the applications using a bottom-up delivery algorithm that allows lower-level applications to mask errors from higher layers. Applications are given the initial context of the I/O request along with the error allowing them to only incur additional overhead when exceptional conditions arise. The storage applications process and transform errors from disparate back-end devices and unify the presentation of these errors to make the volume appear as a single device to the hosts that access it.

Referring to Fig. 21, a schematic block diagram of elements of the architecture shown in the above-described Figures is presented to show the relationship of the layered drivers with the Device Object Graph 410 and its logical volumes in group 412, and the relationship to Backend arrays 420 including physical volumes 422-426 to which the logical volumes are mapped. The layered drivers TCD/TDD 400, Clone 402, Snap 404, Fusion 406 and FlareX 408 are part of layered drivers 123 discussed above with reference to Fig. 6. Generally each driver presents volumes to the layer above it. For example the FlareX driver imports storage extents from the backend array and presents the identity of such extents to layered drivers above, e.g. Fusion, and Clone drivers.

Regarding nomenclature used for presenting an example of how this embodiment functions, in the volume group 412, V7 represents a "top level" volume, or one that is capable of being presented to a host. V6 represents a cloning volume. This volume has two children, V3 and V5, where V3 is the source for the clone and V5 is its replica. Similarly V1 and V2 are children of V3 while V4 is a child of V5. It is the children at

the bottommost layer that are presented by the bottommost driver in this example (FlareX) to map the storage extents of the backend array that are of interest.

Referring now to Fig. 22, a Device Graph 430 and a Volume Graph 432 illustrate the relationship between drivers and volumes. The Volume Graph is a data structure that is the same as a known Device Graph in known Clariion architecture. The Volume Graph maps to the Device Graph in order to present information needed for virtualization by the IMPS, and provides a mechanism by which errors can be received and handled by the Device Drivers. In an effort to have the host seamlessly communicate with software components discussed herein in the same way it would if communicating with a data storage system such as an EMC Symmetrix or Clariion. Virtualization Software in the SP accomplishes this goal and also by communicating with software in the IMPS to present information about the volumes consistently, including presentation and management of errors. The Volume Graph provides a mechanism by which an error can be introduced into the Device Graph and presented to the host.

Information extracted from the IMPS through its API includes the type of I/O operation (e.g., read or write), a logical volume target or in this context a virtual target (VT), a virtual LUN (v-LUN), physical target, a physical LUN, block address of a physical device, length of request, and error condition. The error condition is received and that identifies the bottommost affected volume and delivered to the bottommost affected object.

Objects in the Volume Graph have one-to-one congruity with objects in the Device Graph, as follows. At the top layer (V7) the Volume Graph Object G represents a

slice volume. A slice volume represents a "slice" or partition of its child volume. It has only one child volume and presents a (possibly complete) contiguous subset of the data of this child volume. The next layer down (V6) is congruently mapped to element F represents the mirrored volume. V3 represents an aggregated volume consisting of the concatenation of volumes V1 and V2. Volume V5 is a slice volume that presents the data of V4. Slice Volume Objects A, B, and D, are congruent with Device Graph Objects V1, V2, and V4, respectively.

Upon receiving an error, the Bottommost Volume Graph Object will deliver the error to its corresponding Device Graph Object. The Device Graph Object as the owner of the error can decide how to handle, e.g. propagate the error up the layer stack, or deal with it directly and not propagate it. Two goals are carried out in the error presentation: consistent presentation to the host, and masking of errors which involves using independent capabilities of the FabricX architecture to solve problems for the host at the unique middle-layer level of a FabricX instance.

Reference is made below to Figs. 23-25, wherein an example of handling of an error for either consistent presentation to the host or masking of the error while handling the error at the FabricX instance or middle switch level is shown. In the example shown in Fig. 24, Virtual Target (VT) LUN or VT/LUN 450 is loaded with a Volume presentation of a Clone or Clone/Volume 452 and having two Children Volumes, i.e. "Child1" 454 and "Child2" 456. In this example case, Child1 and Child2 are synchronized, Child1 represents storage for one type of Data Storage System, for

example an EMC Symmetrix, and Child2 represents storage for another type of Data Storage System, for example an EMC Clariion.

In handling the error in this example, there are two cases for handling in keeping with the goals discussed above: (a) Case 1 – error is not delivered back to host but rather handled by a Device Object at the FabricX instance level; or (b) Case 2 – error is transformed to have the error personality of FabricX. Each method of handling is explained in the example, continued with the flow-diagrams of Figs. 24 and 25.

Referring to Fig. 24, in step 460, an I/O command in the form of a read arrives at an IMPS for the VT/LUN. In step 462, the IMPS selects Child2 to perform the read. The IMPS forwards the read to the data storage system supporting Child2 in step 464. That data storage system returns an error for the read in step 466. In step 468, the IMPS receives the error and presents the error to the SAL Agent in the IMPS in the preferred embodiment, but one skilled in the art will recognize that it could run elsewhere, including in an SP. In steps 468 and 470, the error is propagated from the SAL Agent running on the IMPS to the SAL CPP running on the SP. The SAL CPP then delivers the error to the Volume Graph Manager. The Volume Graph Manager VGM identifies the bottommost affected volume object and delivers the error to this object in step 474. Connecting step 476 labeled "B" connects with the identically numbered and labeled step on Fig. 25.

Referring to Fig. 25, in step 480, the bottommost object (D) delivers or presents the error and its I/O context to the Volume Graph Object Owner (V4). V4 decides it will not handle the error and indicates to D to propagate the error, which D does up to E,

shown in steps 482 and 484. E presents the error to V5 in step 486. In step 490 V5 decides whether to mask the error (Case 1) and handle it itself at the FabricX level, or to return it to the Host (Case 2). If Case 2 is selected, then the error is transformed to have the error personality of FabricX. In this example, that means that V7 updates the error presented by the storage system and using the SAL Agent to modify the error on the switch via the IMPS API. This in turn causes the error to be returned to the host with modified content, but from the switch or FabricX level.

A system and method has been described for managing one or more data storage networks at a middle-layer level using a new architecture. Having described a preferred embodiment of the present invention, it may occur to skilled artisans to incorporate these concepts into other embodiments. Nevertheless, this invention should not be limited to the disclosed embodiment, but rather only by the spirit and scope of the following claims and their equivalents.